Le génie pour l'industrie

-

SYS863 - Internet Industriel des Objets

Laboratoire 1

À Montréal, le 10/02/2023

Professeur : Lokman Sboui
Elèves : Victor Rios, Théodore Ruf

Buts et objectifs du laboratoire :

Dans ce laboratoire nous allons découvrir la programmation en C sur une Raspberry Pi 3 B équipée d'un bcm2835 et les différentes utilisations possibles dans l'IIoT.

Dans un premier temps, nous allons utiliser le logiciel Geany pour écrire, compiler et tester du code en C, pour ainsi configurer des broches GPIOs et enfin utiliser un thread afin de découvrir l'exécution parallèle de programmes.

De plus, dans ce laboratoire nous nous intéresserons particulièrement à la couche de perception de l'IIoT en réalisant un asservissement de température directement sur la Raspberry Pi équipée d'un capteur de température (thermocouple du type K) et d'une lampe 12 V permettant d'augmenter cette température.

Ainsi, on propose d'implanter deux approches de commandes pour l'asservissement de température :

- Une commande de type thermostat (ON/OFF)
- Une commande de type PID

Nous allons pour ce faire implanter une lecture du thermocouple à l'aide du SPI, seulement nous allons créer cette communication de façon software sans utiliser les bibliothèques proposées. De plus, à l'aide d'une commande PWM nous serons capable de commander l'intensité de la lampe. Enfin, nous pourrons implanter les deux commandes différentes et observer le résultat.

2.1 Partie 1 - Initiation à la programmation pour l'IIoT

2.1.1 Compiler et tester un premier programme en C

Dans cette partie, nous apprenons à lire l'état d'un bouton poussoir au travers d'une méthode appelée *polling ;* utilisée pour ses facilités d'implantation. La boucle "while", est l'endroit où le clignotement de la LED se produit. L'instruction "bcm2835_gpio_write" permet de contrôler l'état désiré de la LED.
Elle s'allume si est renseigné l'état logique "HIGH", et s'éteint si est renseigné l'état logique "LOW". Enfin, la fonction "void bcm2835_delay(int duree)" permet de créer une pause d'une durée désirée en milliseconde. On obtient donc une fréquence de 1 Hertz via deux délais de même durée, soit 500 ms. Toutefois, la pause a pour défaut de figer le programme. Cela peut entraîner des contraintes dans un certain contexte.

En effet, la présence de faibles fréquences comme ici (0.1 hertz, soit une période de 10 secondes) nous met face à un problème de temps réel. Il faut rester appuyer et attendre que les deux DEL changent d'état pour arrêter le programme, soit attendre au moins 5 secondes.

Le *polling* est donc une méthode qui peut s'avérer compromettante, notamment d'un point de vue industriel. Pour des situations d'urgence, elle n'est pas adaptée car n'est pas temps réel.

2.1.2 Compiler et tester un programme utilisant un thread

Dans cette seconde partie, nous faisons aussi clignoter les LEDs, mais cette fois via une nouvelle méthode qui s'appelle *thread.* L'intérêt de celle-ci est qu'il est possible de rendre l'appui des boutons poussoirs **indépendants** du clignotement des LEDs.

- En créant les threads (avec la fonction *pthread_create*) on peut exécuter les tâches, ici le clignotement des LEDs, en parallèle avec le *main*.
- L'appui sur les boutons-poussoir est la condition de sortie immédiate de la boucle while dans le programme principal et cette condition est vérifiée à toutes les millisecondes. La boucle est déclarée après la création de nos deux threads.
- La condition d'arrêt satisfaite, on arrête les threads et attendons leur arrêt total avant d'éteindre les LEDs.

La réponse pour arrêter le programme est donc améliorée car elle est maintenant quasiment instantanée. Peu importe le délai de clignotement des LEDs, on aura une réaction immédiate lors de l'appui sur le bouton, qui pour rappel, est devenu **indépendant** de la fréquence de clignotement choisie grâce aux threads.

Pour illustrer cette notion de "parallèle" une brève définition : avec l'amélioration des processeurs, de nouvelles fonctionnalités furent développées notamment la possibilité d'exécuter des "processus légers" (nommés threads) s'exécutant en "même temps". En fait l'exécution n'est pas simultanée, mais plutôt concurrente. Par exemple, la Figure 1. montre l'exécution de deux threads de façon concurrente par un CPU. On constate que le CPU

exécute un petit bout de l'un et un petit bout de l'autre. Un planificateur décide du moment et de la durée allouée à tous les threads et autres fonctions s'exécutant dans le système. Cela nous donne l'impression que tout se fait en même temps, bien que ce ne soit pas le cas en réalité, car le CPU ne peut faire qu'une tâche à la fois.

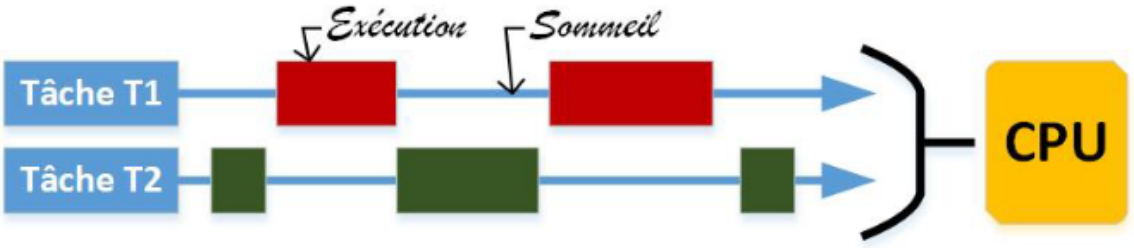

Figure 1. Exécution concurrente de deux threads sur un CPU

2.1.3 Compiler et tester un programme avec délai précis

Dans cette partie, on se propose d'utiliser des délais très précis pour des fréquences élevées. Ainsi, à la place d'utiliser des délais classiques, on peut récupérer le nombre de coups d'horloge depuis le lancement du programme à l'aide de la bibliothèque time.h. En comptabilisant le nombre de coups d'horloge par seconde, on peut obtenir le temps d'exécution d'une section de code. Cela est très intéressant lorsqu'on échantillonne des signaux à des fréquences élevées (1MHz).

Ainsi, on peut définir la fréquence à laquelle notre DEL va clignoter en calculant la demi période des deux phases : allumée et éteinte. Le délai est ainsi ajusté entre le début du code et la fin de son exécution grâce aux fonctions de la librairie time.h et nous permet d'assurer une période de clignotement de la DEL définie et régulière.

On peut retrouver cette partie de code dans le listing *EquipeA_Lab1_Partie1.c* en annexe. On constate dans notre programme que pour une fréquence de clignotement fixé à 10 Hertz, on observe bien 10 clignotements de DEL par seconde : T= 1/f = 1/10 = 0.1s .

2.2 Partie 2 - Lecture de la température sur le MAX31855

2.2.1 L'utilisation du SPI sur le MAX31855

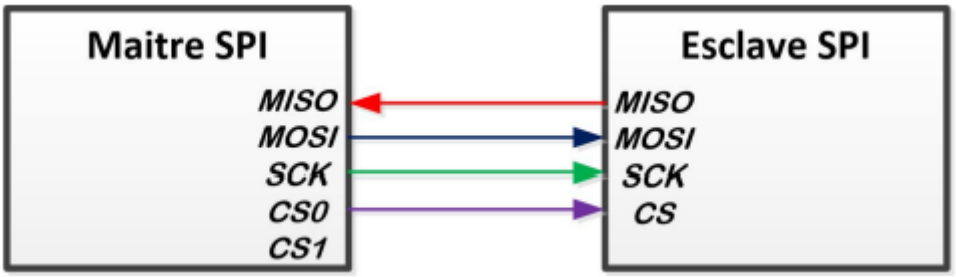

Figure 2. Configuration minimale d'un SPI

Cette partie concerne l'utilisation d'un protocole de communication full-duplex appelé SPI pour Serial Peripherical Interface. Son approche maître/esclave lui permet de faire communiquer des composants sur des canaux différents sans se soucier d'overwritting. Dans un souci d'apprentissage, nous allons programmer cette communication de façon logicielle manuellement. Ainsi, nous pourrons identifier ce qu'il se passe réellement lors d'une telle communication.

Le composant responsable de la récupération de la température est le MAX31855, sa transmission de données par SP implique :

- la descente du signal de sélection d'esclave CS (Chip Select)
- l'envoie de 32 signaux d'horloge (SCK) correspondant aux 32 bits de la trame SPI
- la fin de la transmission par le front montant du signal CS

On peut ainsi reproduire ce fonctionnement de fonctionnement de façon logicielle en jouant sur les états logiques des signaux et en introduisant des délais pour simuler les coups d'horloge.

*Remarque : On peut choisir de travailler avec des fréquences de fonctionnement pour le MAX31855 bien inférieures à 5 MHz sans poser de problèmes. Cependant, pour respecter le Théorème de Shannon (fe > 2 * fmax) avec une fréquence d'acquisition du contrôleur de température de 1 Hz il faut au minimum que la fréquence du MAX31855 soit supérieure à 2 Hz.*

2.2.2 La lecture du signal reçu du MAX31855

On peut observer ci-contre le chronogramme de communication SPI avec le MAX31855 que l'on a utilisé pour construire notre récupération software.

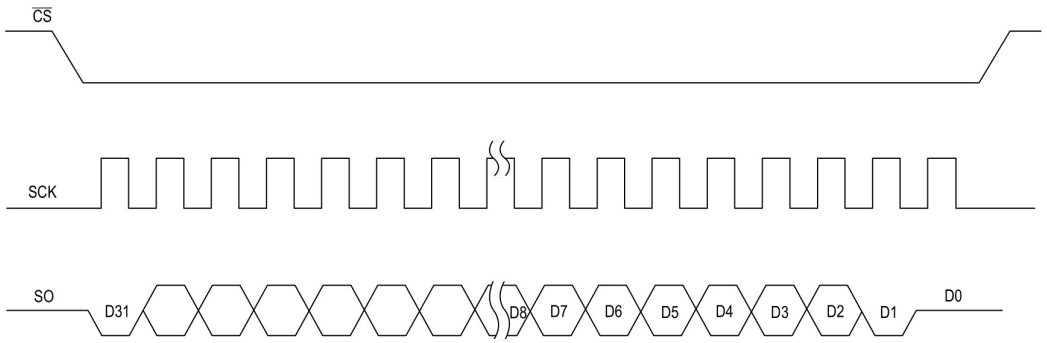

Figure 3. Chronogramme de communication SPI avec le MAX31855

On peut identifier dans la trame les 14 premiers bits (bit D31 à bit D18) du message qui nous intéresserons pour la suite car ils contiennent la température mesurée par le thermocouple.

2.2.3 Lecture et affichage de la température

Le but de cette partie est la conception de la routine qui lit la température du thermocouple. Ainsi, on créera la fonction **lecture_SPI** qui retournera la température mesurée en C°.

Après l'initialisation des GPIO dans cette même fonction, on suivra alors le fonctionnement décrit plus haut c'est-à-dire l'initialisation de la lecture par descente du signal CS avec `bcm2835_gpio_write(CS, LOW).` Ensuite on s'assurera de récupérer 32 bits en passant dans une boucle for qui crée le signal d'horloge SCK (montant/decendant) remplit un **buffer** et décale de 1 bit la valeur précédente à chaque itération pour éviter l'overwritting. Enfin, comme annoncé on utilisera seulement les 18 premiers bits du buffer (MSB), en décalant notre buffer de 18 bits vers la droite, on peut ainsi récupérer la température en castant le buffer en float en extraire la partie décimale :

```
buffer = (buffer >> 18) ;
double temp = (float)buffer/4;
```

On peut enfin récupérer la valeur de cette température en l'affichant dans le terminal et en faisant conjointement clignoter la LED une fois par seconde.

2.3 Partie 3 - Commander une lampe par PWM

2.3.1 L'utilisation du PWM

Cette partie concerne l'utilisation de la modulation de largeur d'impulsions (Pulse Width Modulation – PWM). Cette dernière est une technique de modulation que l'on utilise souvent pour générer des signaux dont la valeur moyenne de la tension peut changer. On le retrouve notamment dans les cas où l'on doit commander l'amplitude moyenne d'une tension dans un moteur électrique (variateur électronique de vitesse).

Dans notre cas, nous allons commander l'intensité lumineuse dans une lampe de 12 volts pour qu'elle soit plus ou moins chaude.

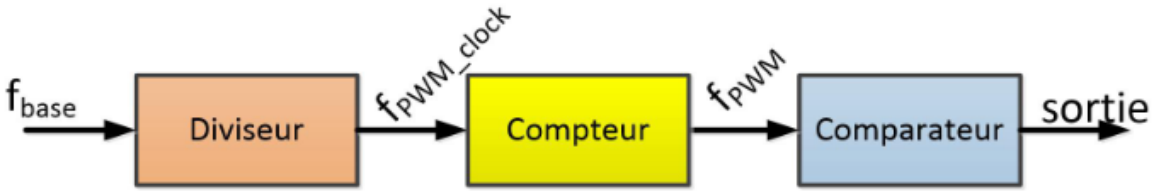

Figure 4. Schéma bloc d'une sortie PWM

2.3.2 Commande de la lampe

Avec une f_{base} à 19,2 MHz, la configuration de notre signal PWM est la suivante :

- Le PWM génère un signal de 440 Hz (f_{pwm})
- Le diviseur est défini à 128
- On trouve un "range" à 150

La fonction intensity qui sera appelée dans la fonction clignote() change l'intensité de la lampe de 12 Volts entre 0% et 100% puis de 100% vers 0% avec un pas de 1% un délai de 10ms entre deux changements d'intensité. Elle actualise la valeur du duty cycle avec "bcm2835_pwm_set_data(0,*commande*)" où *commande* est le paramètre ajusté à la valeur correspondant au duty cycle désiré. Tant qu'il est inférieur à 150, on l'incrémente jusqu'à 100%. Le cas inverse, on le décrémente jusqu'à 0%.

A la fin, elle s'assure que la commande soit nulle. Avec un range de 150, une incrémentation de 1.5, équivaut à une incrémentation de 1%.

2.4 Partie 4 - La commande du thermostat

2.4.1 La commande de type thermostat (Tout ou rien - ON/OFF)

On se propose ici d'implanter la commande de type thermostat (ON/OFF). Cette commande des plus basiques repose sur un système tout ou rien qui prend en entrée une consigne et une température mesurée et qui demande l'allumage de la lampe au maximum ("duty cycle" du PWM à 100%) si mesure < consigne ou au contraire demande son extinction (PWM à 0%) si mesure >= consigne.

Pour ce faire on créera une nouvelle fonction commande_Thermo qui retournera alors la commande envoyé au PWM entre 0 et range défini plus haut. On peut alors directement appeler la fonction lecture_SPI dans les paramètres de cette fonction pour tester son fonctionnement.

Ce genre de commande est très rudimentaire, peu précise mais peut avoir une utilité en terme de consommation pour des systèmes qui ne requièrent pas d'approche particulière de la valeur consigne.

2.4.2 La commande de type PID

Le régulateur PID est une commande largement utilisée dans de nombreux domaines nécessitant des systèmes régulés en boucle fermée. On l'utilise pour maintenir une valeur à un niveau souhaité, ici la température en jouant sur une commande, ici le duty cycle du PWM de la lampe.

Pour décrire brièvement son fonctionnement, ce régulateur utilise les erreurs entre la valeur souhaitée et la valeur réelle pour calculer la correction à apporter au système. La correction est déterminée en combinant trois termes : la proportion de l'erreur actuelle (proportionnel), l'accumulation de l'erreur au fil du temps (intégral) et la vitesse de changement de l'erreur (dérivée).

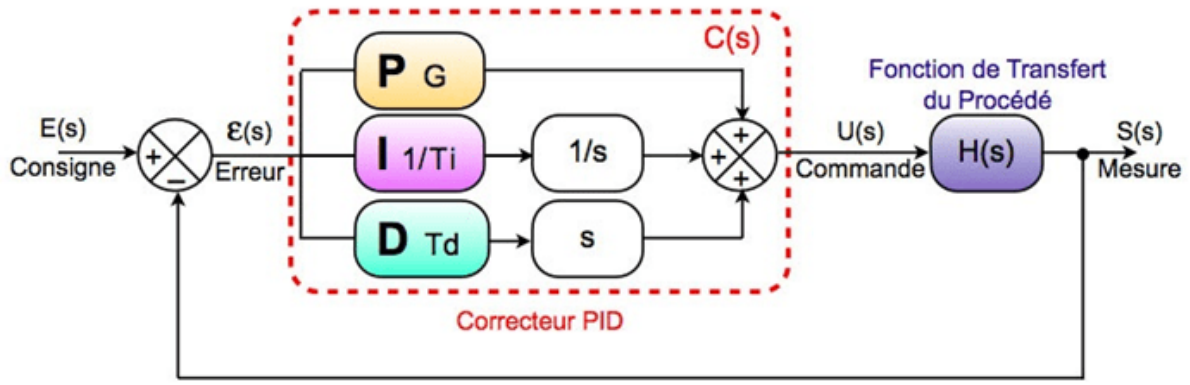

Figure 5. Régulateur PID sur un système en boucle fermée

Pour déployer cette commande on implémente sa forme discrète, dite forme vitesse, et une saturation de la commande PWM tel que : si commande < 0 => commande = 0 et si commande > commande_max => commande = commande_max.

En ce qui concerne l'implémentation de la fonction commande_PID, on écrit simplement en C la forme discrète du régulateur PID. De plus, la fonction prendra en entrée : consigne, mesure et les 3 gains PID en retournant la commande à envoyer. Après quelques tentatives, il s'avère que les gains : Kp = 20, Ki = 20 et Kd = 1, donne une commande dynamique avec un peu de rebond mais un temps de réponse assez bas.

Enfin, dans le thread qui exécute notre boucle fermée, on peut passer directement en paramètre de commande_PID la fonction lecture_SPI qui donne la température mesurée. On peut alors afficher commande et consigne dans le terminal et envoyer la commande (tmp pour tampon dans le listing) récupéré par commande_PID directement dans la fonction intensity qui modifiera le duty cycle de la lampe en conséquence. On peut retrouver cette démarche dans le listing en annexe *EquipeA_Lab1_Partie4.c*.

On constate alors une commande smooth qui adapte la valeur de la température de façon progressive et qui tend à devenir constante bien qu'en réalité on observe qu'elle oscille autour de la valeur consigne. Cette commande est très efficace pour atteindre un équilibre entre une réponse rapide aux perturbations, une élimination des écarts persistants et une anticipation des perturbations futures pour maintenir la valeur mesurée à la valeur désirée.

<u>Bonus</u> :

Nous avons tous les deux apprécié la richesse de ce laboratoire, que ce soit la découverte des threads, l'implémentation software SPI ou l'implémentation discrète du PID. On a pu découvrir les possibilités de la Raspberry. Cependant, on a bloqué pas mal de temps sur des opérations binaires pour le SPI ce qui n'était pas vraiment intéressant dans le cadre de l'IIoT. Une piste pourrait être de faciliter cette partie software par l'introduction d'un tutoriel, (comme dans le Lab2 avec le tutoriel texte sur le graphique excel).